Tugas Besar 2 IF3170 Inteligensi Buatan

Implementasi Algoritma Pembelajaran Mesin

Diajukan untuk memenuhi tugas mata kuliah IF3170 Inteligensi Buatan

Disusun oleh Kelompok 26:

Maggie Zeta RS	13521117
Shafiq Irfansyah	13522003
Devinzen	13522064
Rayendra Althaf TN	13522107
Chelvadinda	13522154

PROGRAM STUDI SARJANA TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

DAFTAR ISI

BAB 1

Penjelasan Implementasi Program

1. Implementasi Algoritma KNN

K-Nearest Neighbor (KNN) adalah algoritma berbasis *instance* yang menggunakan pendekatan *lazy learning*. Algoritma ini menyimpan semua data train dan melakukan prediksi data *test* berdasarkan kemiripan (*distance*) dengan *K-Nearest Neighbor*.
Implementasi:

1. *Distance*
   Kami menggunakan *Minkowski Distance*:
   - p=1: *Manhattan*
   - p=2: *Euclidean*

   Rumus *Minkowski Distance*:

$$d(A, B) = \left(\sum_{i=1}^{n} \left| A_i - B_i \right|^p \right)^{\frac{1}{p}}$$

2. Prediksi
   - Menghitung jarak semua data *test* ke data *train*
   - Menggunakan mayoritas *class* dari *K-Nearest Neighbor* menggunakan `Counter`
3. Implementasi *From Scratch*
   Kami menggunakan *class* `KNN_FromScratch` yang menyimpan data *train* dan melakukan prediksi *class* data *test*.
   - `fit()`: untuk menyimpan data *train*
   - `predict()`: untuk memprediksi *class* data *test*

```
import numpy as np
from collections import Counter

def distance(A, B, p):
    # A dan B array point, p parameter (lihat
__init__)
    return np.sum(np.abs(A - B) ** p) ** (1 / p)

class KNN_FromScratch:
    def __init__(self, n_neighbors: int, p: float):
        # p: parameter angka untuk menghitung
distance Minkowski
        # 1 untuk manhattan, 2 untuk euclidean,
minkowski terserah mau masukin angka berapa (jangan
yg aneh aneh)
        self.n_neighbors = n_neighbors
```

```
        self.p = p

    def fit(self, X_train, Y_train):
        # asumsi scaling dan encoding sudah di
notebook
        # fungsi ini cuma simpan training data
        self.X_train = X_train
        self.Y_train = Y_train
        return

    def predict_instance(self, instance):
        distances = [distance(i, instance, self.p)
for i in self.X_train]
        nearest_neighbors_index =
np.argsort(distances)[:self.n_neighbors] # argsort yg
direturn indexnya
        nearest_neighbors  = [self.Y_train[i] for i
in nearest_neighbors_index]
        return
Counter(nearest_neighbors).most_common(1)[0][0] #
cari yg paling banyak

    def predict(self, X_test):
        # X_test itu semua data test
        return [self.predict_instance(instance) for
instance in X_test]
```

2. Implementasi Algoritma Naive Bayes

Gaussian Naive Bayes adalah bagian dari algoritma *Naive Bayes* yang mengasumsikan bahwa fitur-fitur dalam *dataset* sudah mengikuti distribusi *Gaussian* atau normal. Implementasi menggunakan konsep *probability posterior* berdasarkan *Bayes Theorem*.
Implementasi:

1. *Training Model* (`fit`)
   - $P(v_j)$: *Prior probability*

     $P(v_j) = \frac{Jumlah\ data\ di\ kelas\ v_j}{Jumah\ total\ data}$
   - Distribusi *Gaussian*:

     $mean = \frac{\Sigma x}{N},\ std = \sqrt{\frac{\Sigma(x-mean)^2}{N}}$

2. Menghitung *Likelihood* (`_calculate_likelihood`)

$$P(x|v_j) = \frac{1}{\sqrt{2\pi \cdot \sigma^2}} \cdot exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

3. *Posterior Probability* (`_calculate_posterior`)

$$P(v_j|X) = P(v_j) \cdot P(x_1|v_j) \cdot P(x_2|v_j) \ldots P(x_n|v_j)$$

   - $P(v_j)$: *prior class*
   - $P(x_i|v_j)$: *likelihood*

4. Prediksi
   - Menghitung probabilitas posterior untuk semua *class*
   - Memilih *Class* dengan posterior paling tinggi sebagai prediksi
5. Implementasi
   - Model dapat disimpan menggunakan *pickle*

```
import numpy as np
import pandas as pd
from collections import defaultdict
import pickle

class GaussianNaiveBayes:
    def __init__(self):
        self.priors = {}
        self.mean_std = defaultdict(dict)
        self.classes = None

    def fit(self, X, y):
        # Pastikan y berbentuk array numpy
        y = np.array(y)
        self.classes = np.unique(y)

        for c in self.classes:
            # Ambil subset data untuk setiap kelas
            X_c = X[y == c]

            # Hitung prior P(vj) untuk kelas c
            self.priors[c] = len(X_c) / len(y)

            # Hitung mean dan std dev untuk setiap
fitur di kelas c
            self.mean_std[c]['mean'] =
X_c.mean(axis=0)
            self.mean_std[c]['std'] = X_c.std(axis=0)

    def _calculate_likelihood(self, x, mean, std):
        # Menghitung distribusi Gaussian
        exponent = np.exp(-((x - mean) ** 2) / (2 *
```

```
std ** 2))
        return (1 / (np.sqrt(2 * np.pi) * std)) *
exponent

    def _calculate_posterior(self, x):
        posteriors = {}

        for c in self.classes:
            # Ambil prior P(vj)
            prior = self.priors[c]

            # Ambil mean dan std dev untuk kelas c
            mean = self.mean_std[c]['mean']
            std = self.mean_std[c]['std']

            # Hitung likelihood P(a1 | vj) * P(a2 |
vj) * ... * P(an | vj)
            likelihood =
np.prod(self._calculate_likelihood(x, mean, std))

            # Hitung posterior P(vj | a1, a2, ...,
an)
            posteriors[c] = prior * likelihood

        return posteriors

    def predict(self, X):
        # Konversi X ke array numpy jika berbentuk
DataFrame
        X = np.array(X)
        predictions = []
        for x in X:
            # Hitung posterior untuk setiap kelas
            posteriors = self._calculate_posterior(x)
            # Pilih kelas dengan posterior tertinggi
            predictions.append(max(posteriors,
key=posteriors.get))
        return predictions

    def save_model(self, file_name):
        """Menyimpan model ke file."""
        with open(file_name, 'wb') as file:
            pickle.dump(self, file)
        print(f"Model saved to {file_name}")

    @staticmethod
```

```
    def load_model(file_name):
        """Memuat model dari file."""
        with open(file_name, 'rb') as file:
            model = pickle.load(file)
        print(f"Model loaded from {file_name}")
        return model
```

3. Implementasi Algoritma ID3

ID3 (*Iterative Dichotomiser 3)* adalah algoritma *decision tree learning* yang menggunakan *information gain* untuk membangun *tree* berdasarkan atribut-atribut *dataset*.

Implementasi:

1. Menghitung *Entropy*
   Mengukur ketidakpastian *class* dalam *dataset*:
   $$H(S) = -\sum_{i=1}^{n} p_i log_2(p_i)$$
2. *Information Gain*
   Menentukan fitur paling baik untuk memisahkan data:
   $$IG(S, A) = H(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} H(S_v)$$
3. Implementasi
   - Fungsi `information_gain`: untuk mencari *threshold* paling baik
   - Membentuk pohon dengan menghitung *entropy* dataset dan membagi ke *left* dan *right node* dan rekursif hingga semua *instance* memiliki *class* yang sama atau mencapai `max_depth`

```
import pickle
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

# Ngitung entropy
def entropy(y):
    classes, counts = np.unique(y,
return_counts=True)
    probabilities = counts / len(y)
    return -np.sum(probabilities *
np.log2(probabilities))
```

```
# Menghitung information gain untuk sebuah fitur dan
threshold tertentu
def information_gain(data, feature, target):
    total_entropy = entropy(target)

    # Inisialisasi threshold
    best_threshold = None
    best_gain = -1

    # Iterasi pada nilai unik dalam fitur untuk
mencari threshold terbaik
    unique_values = np.unique(data[feature])
    for threshold in unique_values:
        # Membagi data menjadi dua grup berdasarkan
threshold
        left_split = target[data[feature] <=
threshold]
        right_split = target[data[feature] >
threshold]

        # Menghitung entropy dari masing-masing grup
        weight_left = len(left_split) / len(target)
        weight_right = len(right_split) / len(target)
        split_entropy = (weight_left *
entropy(left_split)) + (weight_right *
entropy(right_split))

        # Menghitung information gain
        gain = total_entropy - split_entropy

        # Memperbarui threshold terbaik jika gain
lebih baik
        if gain > best_gain:
            best_gain = gain
            best_threshold = threshold

    return best_gain, best_threshold


class ID3Algorithm:
    def __init__(self, max_depth=None):
        self.tree = None
        self.max_depth = max_depth
        self.message = "ID3 Numeric Classifier"

    # Membangun decision tree secara rekursif
```

```
    def fitter(self, X, y, depth=0):
        # Jika semua data memiliki kelas yang
sama/kedalaman maksimum tercapai, bikin node daun
        if len(np.unique(y)) == 1 or (self.max_depth
is not None and depth >= self.max_depth):
            return np.unique(y)[0]  # Node daun
dengan kelas dominan

        # Inisialisasi fitur, threshold, dan gain
terbaik
        best_feature = None
        best_threshold = None
        best_gain = -1

        # Cari fitur dan threshold terbaik dari
information gain
        for feature in X.columns:
            gain, threshold = information_gain(X,
feature, y)
            if gain > best_gain:
                best_gain = gain
                best_feature = feature
                best_threshold = threshold

        # Kalo gada gain, buat node daun dengan kelas
dominan
        if best_gain == 0:
            return np.unique(y)[0]

        # Bikin node internal dengan fitur dan
threshold terbgs
        tree = {
            'feature': best_feature,
            'threshold': best_threshold,
            'left': None,
            'right': None
        }

        # Rekursif ke cabang kiri dan kanan
        left_indices = X[best_feature] <=
best_threshold
        right_indices = X[best_feature] >
best_threshold

        tree['left'] = self.fitter(X[left_indices],
y[left_indices], depth + 1)
```

```
        tree['right'] = self.fitter(X[right_indices],
y[right_indices], depth + 1)

        # self.tree = tree
        return tree

    def fit(self, X, y):
        self.tree = self.fitter(X, y)

    # Prediksi kelas untuk satu instance menggunakan
decision tree
    def predict_instance(self, instance, tree):
        # Jika tree adalah node daun, kembalikan
kelasnya
        if not isinstance(tree, dict):
            return tree

        # Arahkan ke cabang kiri atau kanan
berdasarkan nilai threshold
        feature = tree['feature']
        threshold = tree['threshold']

        if instance[feature] <= threshold:
            return self.predict_instance(instance,
tree['left'])
        else:
            return self.predict_instance(instance,
tree['right'])

    # Prediksi kelas untuk semua data (hasilnya
adalah list dengan kelasnnya)
    def predict(self, X):
        return [self.predict_instance(row, self.tree)
for _, row in X.iterrows()]

    # Simpan model
    def save_model(self, file_name):
        """Save the model to a file."""
        with open(file_name, 'wb') as file:
            pickle.dump(self, file)
        print(f"Model saved to {file_name}")

    # Load model
    @staticmethod
    def load_model(file_name):
        """Load the model from a file."""
```

```
        with open(file_name, 'rb') as file:
            model = pickle.load(file)
        print(f"Model loaded from {file_name}")
        return model
```

BAB 2

Tahap *Cleaning* dan *Preprocessing*

1. *Data Cleaning*

- *Handling Missing Data*

  Missing values pada data numerik di-*handle* dengan imputasi mean atau median karena mean dan median efektif untuk data numerik yang tidak memiliki banyak *outlier* sedangkan fitur kategori menggunakan mode karena untuk mempertahankan nilai yang paling umum. Implementasi kelas `FeatureImputer` untuk meng*handle* kolom dengan nilai kosong yang melebihi batas tertentu.
- *Handling Outliers*

  Untuk memotong nilai *outlier* di luar batas yang ditentukan digunakan teknik *clipping* dengan IQR (*Interquartile Range*) karena *outlier* dapat merusak performa algoritma yang *distance based* seperti KNN dan *clipping* untuk mempertahankan data asli sambil membatasi akibat dari *outlier* tersebut.
- *Remove Duplicates*

  Data yang duplikat dihapus berdasarkan kolom id dengan menggunakan `drop_duplicates` karena duplikat bisa menyebabkan bias dan mempengaruhi performa model. Implementasi kelas `DuplicateImputer` untuk mengidentifikasi dan menghapus data yang duplikat.

2. *Data Preprocessing*

- *Scaling*

  Menggunakan *Min-Max Scaling* untuk meratakan *range* nilai numerik ke dalam skala [0,1] karena algoritma KNN dan *Naive Bayes* sensitif terhadap skala data dan *Scaling* memastikan semua fitur memiliki kontribusi yang seimbang. Implementasi kelas `FeatureScaler` untuk menghitung nilai minimum dan maksimum dari semua fitur yang numerik.
- *Encoding*

  Menggunakan *One-Hot Encoding* di fitur kategori untuk mengubah menjadi format numerik karena algoritma *machine learning* secara umum bekerja dengan data yang numerik dan *One-Hot Encoding* mempertahankan nilai unik dari fitur kategori tanpa mengasumsikan urutan. Implementasi kelas `FeatureEncoder` untuk meng*encode* fitur *categorical*.
- *Handling Imbalanced Data*

  Menggunakan SMOTE (*Synthetic Minority Oversampling Technique*) untuk menyeimbangkan distribusi kelas karena distribusi kelas pada *dataset* tidak seimbang, seperti misalnya kelas `normal` dan `attack` berbeda signifikan dan

SMOTE efektif untuk menambah sampel kelas yang minoritas tanpa kehilangan informasi. Implementasi kelas `ImbalancedDataHandler` untuk melakukan *oversampling* menggunakan SMOTE.

- *Dimensionality Reduction*
  Menggunakan PCA (*Principal Component Analysis*) untuk mengurangi dimensi fitur sambil mempertahankan variasi data karena *dataset* yang punya fitur terlalu banyak memiliki risiko *overfitting* dan PCA membantu mengurangi fitur redundan dan menyederhanakan data. Implementasi kelas `PCAImputer` untuk melakukan reduksi dimensi dengan menghitung komponen utama.

3. *Pipeline Preprocessing*

Pipeline mengombinasikan semua langkah di atas untuk memastikan proses *cleaning* dan transformasi data berjalan dengan baik. *Pipeline* memastikan data siap untuk digunakan sebagai model *machine learning*.
Langkah-langkah *pipeline*:

1. *Outlier Handling* (`OutlierImputer`)
2. *Missing Value Imputation* (`FeatureImputer`)
3. *Duplicate Removal* (`DuplicateImputer`)
4. *Scaling dan Encoding* (`FeatureScaler` dan `FeatureEncoder`)
5. *Dimensionality Reduction* (`PCAImputer`)
6. *Scaling* (`FeatureScaler`)
7. *Handling Imbalanced Data* (`SMOTE`)

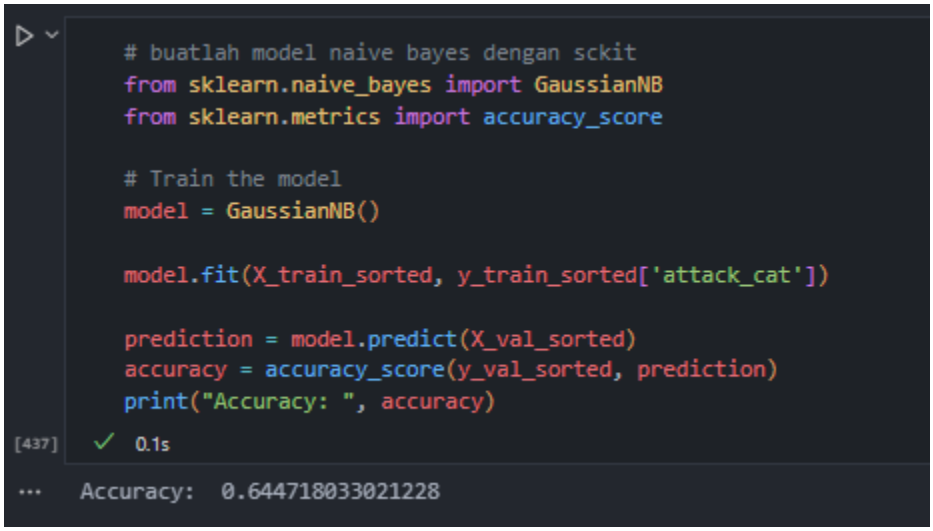
Dengan menangani *missing values*, *outliers*, dan duplikat bisa menjaga kebersihan data dan integritas *dataset*. Selain itu, *scaling* membuat algoritma jarak bekerja lebih baik. *Dimensionality reduction* membantu mengurangi kekompleksitasan model dan kemudian dilakukan *scaling* lagi untuk menyesuaikan hasil PCA. *Handling imbalanced data* ditujukan untuk membuat performa model seimbang untuk semua kelas. Sehingga algoritma seperti KNN, ID3, dan *Naive Bayes* dapat bekerja optimal dan menghasilkan prediksi yang akurat.

BAB 3

Hasil

1. Hasil Algoritma Naive Bayes

Hasil implementasi algoritma Naive Bayes menunjukkan bahwa model berhasil mencapai akurasi sebesar 39.04% setelah dilatih menggunakan data yang sudah diurutkan. Proses prediksi dilakukan pada data validasi, dan hasilnya dibandingkan dengan label sebenarnya. Akurasi ini menunjukkan bahwa model mampu mengenali sebagian pola dalam data, meskipun performanya masih terbatas. Hal ini kemungkinan terjadi karena karakteristik data yang lebih kompleks dibandingkan dengan asumsi sederhana dari algoritma Naive Bayes yang menggunakan distribusi Gaussian.



jika dibandingkan dengan sckit terlihat berbeda jauh, hal ini dimungkinkan perbedaan pendekatan algoritma yang digunakan.

2. Hasil Algoritma ID3

Pada implementasi algoritma ID3, proses feature selection dilakukan dengan menghitung information gain menggunakan metode binary split. Metode ini dipilih untuk meningkatkan efisiensi waktu perhitungan. Namun, penggunaan binary split memiliki keterbatasan, di mana tidak semua kelas pada target fitur dapat tercapai. Hal ini disebabkan oleh kedalaman pohon yang dibatasi hanya sampai 3, akibat penerapan PCA yang mereduksi fitur menjadi 3 saja. Dalam kondisi terbaik, jumlah kelas yang bisa dicapai adalah $2^3 = 8$, sedangkan jumlah kelas sebenarnya pada target attack_cat adalah 10.

Sempat dipertimbangkan untuk mengganti metode split menjadi quartile split (pembagian menjadi 4 bagian) agar pembagian data lebih optimal. Namun, metode ini membutuhkan waktu komputasi yang sangat lama sehingga dianggap tidak dapat diimplementasikan secara efisien. Setelah validasi dilakukan dengan perbandingan data train dan validation sebesar 2:1, hasil evaluasi menunjukkan bahwa

3. Hasil Algoritma KNN

Pada implentasi KNN, perhitungan terlalu lama jadi pada tangkapan layar di bawah baru memakai 200 data.

```
from KNN import KNN_FromScratch
from sklearn.metrics import accuracy_score
x_train_final = x_train_final.iloc[:200]
y_train_final = y_train_final.iloc[:200]
x_val_final = x_val_final.iloc[:200]
y_val_final = y_val_final.iloc[:200]
model = KNN_FromScratch(5, 2)
model.fit(x_train_final, y_train_final)
predictions = model.predict(x_val_final)
accuracy = accuracy_score(y_val_final, predictions)
print("Accuracy (Home):", accuracy)

model_scikit = KNeighborsClassifier(n_neighbors=5, p=2)
model_scikit.fit(x_train_final, y_train_final)
predictions_scikit = model_scikit.predict(x_val_final)
accuracy_scikit = accuracy_score(y_val_final, predictions_scikit)
print("Accuracy (Scikit):", accuracy_scikit)
```

✓ 0.7s

```
Accuracy (Home): 0.675
Accuracy (Scikit): 0.67
```

KONTRIBUSI

NIM	Nama	Tugas
13521117	Maggie Zeta RS	Laporan, Testing
13522003	Shafiq Irfansyah	ID3, Naive Bayes, Testing
13522064	Devinzen	KNN, Testing
13522107	Rayendra Althaf TN	Preprocessing, Testing
13522154	Chelvadinda	Naive Bayes, Testing

REFERENSI

https://cdn-edunex.itb.ac.id/53145-Artificial-Intelligence-Parallel-Class/210071-Supervised-Learning/90133-Supervised-Learning/1699250331293_IF3170_Materi09_Seg01_AI-kNN.pdf
https://cdn-edunex.itb.ac.id/53145-Artificial-Intelligence-Parallel-Class/210071-Supervised-Learning/90133-Supervised-Learning/1699250380758_IF3170_Materi09_Seg02_AI-NaiveBayes.pdf
https://cdn-edunex.itb.ac.id/53145-Artificial-Intelligence-Parallel-Class/210071-Supervised-Learning/90133-Supervised-Learning/1699250430397_IF3170_Materi09_Seg03_AI-PredictionMeasurement.pdf
https://cdn-edunex.itb.ac.id/64464-Artificial-Intelligence-Parent-Class/297373-Data-Preparation/1730818790714_IF-3170-Data-Preparation.pdf